12-编程范式：明明写的是Java，为什么被人说成了C代码？

你好！我是郑晔。

在上一个小模块，我给你讲了程序设计语言，帮助你重新审视一下自己最熟悉的日常工具。但是，使用程序设计语言是每个程序员都能做到的，可写出的程序却是千差万别的。这一讲，我们就来看看这些差异到底是怎样造成的。

在开始之前，我先给你讲一个小故事。

在一次代码评审中，小李兴致勃勃地给大家讲解自己用心编写的一段代码。这段代码不仅实现了业务功能，还考虑了许多异常场景。所以，面对同事们提出的各种问题，小李能够应对自如。

在讲解的过程中，小李看到同事们纷纷点头赞许，心中不由得生出一丝骄傲：我终于写出一段拿得出手的代码了！讲解完毕，久久未曾发言的技术负责人老赵站了起来：“小李啊！你这段代码从功能上来说，考虑得已经很全面了，这段时间你确实进步很大啊！”

要知道，老赵的功力之深是全公司人所共知的。能得到老赵的肯定，对小李来说，那简直是莫大的荣耀。还没等小李窃喜的劲过去，老赵接着说了，“但是啊，写代码不能只考虑功能，你看你这代码写的，虽然用的是Java，但写出来的简直就是C代码。”

正在兴头上的小李仿佛被人当头泼了一盆冷水，我用的是Java啊！一门正经八百的面向对象程序设计语言，咋就被说成写的是C代码了呢？

“你看啊！所有的代码都是把字段取出来计算，然后，再塞回去。各种不同层面的业务计算混在一起，将来有一点调整，所有的代码都得跟着变。”老赵很不客气地说。还没缓过神来的小李虽然想辩解，但他知道老赵说得是一针见血，指出的问题让人无法反驳。

在实际的开发过程中，有不少人遇到过类似的问题。老赵的意思并不是小李的代码真就成了C代码，而是说用Java写的代码应该有Java的风格，而小李的代码却处处体现着C的风格。

那这里所谓代码的风格到底是什么呢？它就是编程范式。

编程范式

编程范式（Programming paradigm），指的是程序的编写模式。使用了什么编程范式，通常意味着，你主要使用的是什么样的代码结构。从设计的角度说，编程范式决定了你在设计的时候，可以使用的元素有哪些。

现在主流的编程范式主要有三种：

- 结构化编程（structured programming）；
- 面向对象编程（object-oriented programming）；
- 函数式编程（functional programming）。

结构化编程，是大部分程序员最熟悉的编程范式，它通过一些结构化的控制结构进行程序的构建。你最熟悉

的控制结构应该就是if/else这样的选择结构和do/while这样的循环结构了。

结构化编程是最早普及的编程范式，现在最典型的结构化编程语言是C语言。C语言控制结构的影响极其深远，成为了很多程序设计语言的基础。

面向对象编程，是现在最主流的编程范式，它的核心概念就是对象。用面向对象风格写出的程序，本质上就是一堆对象之间的交互。面向对象编程给我们提供了一种管理程序复杂性的方式，其中最重要的概念就是多态（polymorphism）。

现在主流的程序设计语言几乎都提供面向对象编程能力，其中最典型的代表当属Java。

函数式编程，是近些年重新崛起的编程范式。顾名思义，它的核心概念是函数。但是，它的函数来自于数学里面的函数，所以，和我们常规理解的函数有一个极大的不同：不变性。也就是说，一个符号一旦创建就不再改变。

函数式编程的代表性语言应该是LISP。我们在第8讲曾经提到过它。之所以要把这位老祖宗搬出来，因为确实还没有哪门函数式编程语言能够完全独霸一方。

编程范式不仅仅是提供了一个个的概念，更重要的是，它对程序员的能力施加了约束。

- 结构化编程，限制使用goto语句，它是对程序控制权的**直接**转移施加了约束。
- 面向对象编程，限制使用函数指针，它是对程序控制权的**间接**转移施加了约束。
- 函数式编程，限制使用赋值语句，它是对程序中的**赋值**施加了约束。

之后讲到具体的编程范式时，我们再来展开讨论，这些约束到底是什么意思。

与其说这些编程范式是告诉你如何编写程序，倒不如说它们告诉你**不要**怎样做。理解这一点，你才算是真正理解了这些编程范式。

如果你去搜索编程范式的概念，你可能会找到更多的编程范式，比如，逻辑式编程，典型的代表是Prolog语言。但这些编程范式的影响力和受众面都相当有限。如果你想扩展自己的知识面，可以去了解一下。

多范式编程

从道理上讲，编程范式与具体语言的关系不大，这就好比你的思考与用什么语言表达是无关的。但在实际情况中，每一种语言都有自己的主流编程范式。比如，C语言主要是结构化编程，而 Java主要是面向对象编程。

不过，虽然每种语言都有自己的主流编程范式，但丝毫不妨碍程序员们在学习多种编程范式之后，打破“次元壁”，将不同编程范式中的优秀元素吸纳进来。这里的重点是“优秀”，而非“所有”。

举个例子，在Linux的设计中，有一个虚拟文件系统（Virtual File System，简称 VFS）的概念，你可以把它理解成一个文件系统的接口。在所有的接口中，其中最主要的是file_operations，它就对应着我们熟悉的各种文件操作。

下面是这个结构的定义，这个结构很长，我从中截取了一些我们最熟悉的操作：

```
struct file_operations {
  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
  int (*open) (struct inode *, struct file *);
  int (*flush) (struct file *, fl_owner_t id);
  int (*release) (struct inode *, struct file *);
  ...
}
```

如果你要开发一个自己的文件系统，只需要把支持的接口对应着实现一遍，也就是给这个结构体的字段赋值。

我们换个角度看，这个结构体主要的字段都是函数指针，文件系统展现的行为与这些函数的赋值息息相关。只要给这个结构体的字段赋值成不同的参数，也就是把不同的函数关联上，这个文件系统就有了不同的行为。如果熟悉面向对象编程，你会发现，这不就是多态吗？

C是一门典型的结构化编程语言，而VFS的设计展现出来的却是面向对象编程的特点，编程范式的“次元壁”在这里被打破了。

事实上，类似的设计还有很多，比如，Java里有一个著名的基础库，Google出的Guava。它里面就提供了函数式编程的基础设施。在Java 8之前，Java在语法上并不支持函数式编程，但这并不妨碍我们通过类模拟出函数。

配合着Guava提供的基础设施，我很早就开始把函数式编程的方式运用在Java中了。同样，C++有一个functor的概念，也就是函数对象，通过重载 () 这个运算符，让对象模拟函数的行为。

无论是在以结构化编程为主的语言中引入面向对象编程，还是在面向对象为主的语言中引入函数式编程，在一个程序中应用多种编程范式已经成为了一个越来越明显的趋势。

不仅仅是在设计中，现在越来越多的程序设计语言开始将不同编程范式的内容融合起来。Java从Java 8开始引入了Lambda语法，现在我们可以更优雅地写出函数式编程的代码了。同样，C++ 11开始，语法上也开始支持Lambda了。

之所以多范式编程会越来越多，是因为我们的关注点是做出好的设计，写出更容易维护的代码，所以，我们会尝试着把不同编程风格中优秀的元素放在一起。比如，**我们采用面向对象来组织程序，而在每个类具体的接口设计上，采用函数式编程的风格，在具体的实现中使用结构化编程提供的控制结构**。

让我们回过头，看看开篇故事小李的委屈吧！老赵之所以批评小李，关键点就是小李并没有把各种编程范式中优秀的元素放到一起。Java是提供对面向对象的支持，面向对象的强项在于程序的值，它归功的设计元素应该是对象，程序应该是靠对象的组合来完成，而小李去把它写成了平铺直叙的结构化代码，这当然是不值得鼓励的。

对于今天的程序员来说，**学习不同的编程范式，将不同编程范式中的优秀元素应用在我们日常的软件设计之中，已经由原来的可选项变成了现在的必选项**。否则，你即便拥有强大的现代化武器，也只能用作古代的冷兵器。

总结时刻

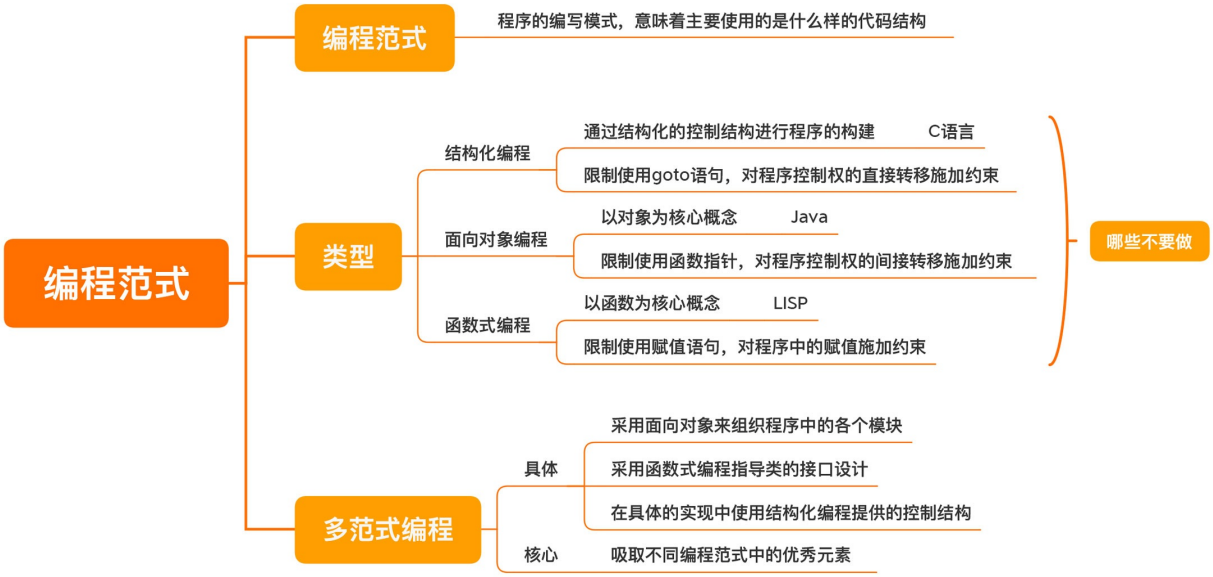
今天，我们今天讨论了编程范式。编程范式指的是程序的编写模式。现在主流的编程范式主要有三种：结构化编程、面向对象编程和函数式编程。编程范式对程序员的能力施加了约束，理解编程范式的一个关键点在于，**哪些事情不要做**。

从道理上讲，编程范式与具体语言的关系不大，但很多语言都有着自己主流的编程范式。但现在的一个趋势是，打破编程范式的“次元壁”，把不同编程范式中优秀的元素放在一起。

一方面，我们可以通过设计，模拟出其他编程范式中的元素，另一方面，程序设计语言的发展趋势也是要融合不同编程范式中优秀的元素。学习不同的编程范式，已经成为每个程序员的必修课。

在接下来的几讲里，我们就来深入地讨论一下各种编程范式。

如果今天的内容你只能记住一件事，那请记住：**学习不同的编程范式，将其中优秀的元素运用在日常工作中**。



极客时间

思考题

今天我们谈到了编程范式，每个程序员都会有自己特别熟悉的编程范式，但今天我想请你分享一下，你在学习其他编程范式时，给你思想上带来最大冲击的内容是什么。欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- Kǎfkǎ²⁰²⁰ 2020-06-22 00:15:33
  当Bob大叔说出那句，“编程范式本质是从某方面对程序员编程能力的限制和规范”时，真有些振聋发聩[4赞]

  作者回复2020-06-22 10:32:12
  哈哈，你看出我从哪吸收营养了。

- chenzesam 2020-06-22 08:57:12
不单止编程范式对程序员的能力做了限制，编程框架也在开始做这一方面的努力了。努力提高程序员的下限。 [2赞]

作者回复2020-06-22 10:30:34
这个理解是对的，编程框架本来就是让你少和底层细节打交道。

- Demon.Lee 2020-06-22 14:57:10
而在每个类具体的接口设计上，采用函数式编程的风格

老师，这句话具体如何理解，我脑子里出现的是java8里面的@FunctionalInterface，很多接口中的函数入参都是一个个@FunctionalInterface，比如：
public interface Predicate<T> {
default Predicate<T> and(Predicate<? super T> other) {
Objects.requireNonNull(other);
return (t) -> test(t) && other.test(t);
}
...
} [1赞]

作者回复2020-06-22 16:38:38
后面讲到函数式编程的时候，你就会看到我的做法了，简言之，设计可以组合的接口。

- Jxin 2020-06-22 00:52:24
以函数式编程为例。
1.我能理解不变性的价值，毕竟在应对并发场景时我也用cow的模式。但很难接受将cow贯彻到每个函数，本能的觉得浪费。
2.我看得到函数式编程在代码上的简洁（可读性高）。但将功能实现成函数式编程的风格，感觉比较难（也可能是我水平不行，毕竟没有刻意练习），而难本身就是成本。（业务逻辑翻译成功能代码，从易到难：面向过程，面向对象，函数式编程）。
3.虽然我理解鸭子理论。但我就想明确的定义接口。因为，当我作为调用方时，我只想知道意图，而没有实现的接口，显然是比较整洁的。
4.虽然我理解函数是一等公民（单方法接口）的定位。但我就喜欢接口下定义多个方法（行为），因为我认为接口是一类事务共同行为的抽象，那么行为很可能是捆绑出现。比如说，对动物行为做抽象，吃和拉必须一起出现，只有吃没有拉，只有拉没有吃都挺尴尬。 [1赞]

- NIU 2020-06-22 09:22:23
如果有介绍各种需要编程范式发展和应用的资料就好了，比如Objective-C，Swift等等

作者回复2020-06-22 23:00:27
编程范式，我们后面聊。

- 阳仔 2020-06-22 07:36:33
主流的编程语言都有结构化编程，面向对象编程，函数编程。纯粹单一的使用某个编程范式在现代编程语言其实会越来越少，现代语言都是吸收了各种编程范式的优点组合编程

作者回复2020-06-22 10:30:58
没错！